Title

stata.com

bayespredict — Bayesian predictions

Description

`bayespredict` computes Bayesian predictions using current estimation results produced by `bayesmh` with built-in likelihood models or by `bayes: var`, `bayes: xtreg`, `bayes: xtlogit`, `bayes: xtprobit`, `bayes: xtologit`, `bayes: xtoprobit`, `bayes: xtpoisson`, `bayes: xtnbreg`, or `bayes: xtmlogit`. The Bayesian predictions are saved in a separate Stata dataset. Bayesian predictions include simulated outcomes, which are samples from the posterior predictive distribution of the fitted Bayesian model, and their functions. You can also compute posterior summaries of simulated outcomes and store them as new variables in the current dataset.

`bayesreps` generates a random subset of MCMC replicates of simulated outcomes from the entire MCMC sample and stores them as new variables in the current dataset. This command is useful for checking model fit.

`bayespredict` and `bayesreps` require that you first save MCMC results by using the `saving()` option with the `bayesmh` command or the `bayes` prefix.

Quick start

Simulated outcomes

Predictions for the first outcome variable after fitting a two-equation Bayesian model using `bayesmh`

```
bayespredict {_ysim}, saving(prdata)
```

As above, but for the second outcome variable, replacing `prdata.dta` with new prediction results

```
bayespredict {_ysim2}, saving(prdata, replace)
```

Predictions for the first outcome variable and observations 2 through 5

```
bayespredict {_ysim1[2/5]}, saving(prdata, replace)
```

Test statistics for simulated outcomes

Maximums and minimums of simulated outcomes computed over observations for the first outcome variable

```
bayespredict (rmax:@max({_ysim1})) (rmin:@min({_ysim1})),    ///
      saving(prdata, replace)
```

Maximums and minimums of residuals for the second outcome variable

```
bayespredict (rmax:@max({_resid2})) (rmin:@min({_resid2})),  ///
      saving(prdata, replace)
```

Posterior summaries of simulated outcomes

Posterior means for the two outcomes stored in new variables `pmean1` and `pmean2` in the current dataset

```
bayespredict pmean1 pmean2, mean
```

As above, but calculating posterior medians and storing them in new variables `pmedian1` and `pmedian2` specified as a variable stub `pmedian*`

```
bayespredict pmedian*, median
```

95% credible intervals for the second outcome variable `y2`; the lower and upper bounds are stored in `cril2` and `criu2`, respectively

```
bayespredict cril2 criu2, cri outcome(y2)
```

Simulate and save MCMC replicates of simulated outcomes

Generate 10 MCMC replicates for the first outcome in the model, and store them as new variables `y1rep1`, ..., `y1rep10` in the current dataset

```
bayesreps y1rep*, nreps(10)
```

As above, but for the second outcome `y2` and storing the results in new variables `y2rep1`, ..., `y2rep10`

```
bayesreps y2rep*, nreps(10) outcome(y2)
```

Menu

Statistics > Bayesian analysis > Predictions

Syntax

Syntax is presented under the following headings:

Compute predictions

Prediction of selected outcome variables and observations

```
bayespredict ysimspec [ysimspec ...] [if] [in], saving(filespec) [simopts]
```

Functions of simulated outcomes, expected values, and residuals

```
bayespredict (funcspec) [(funcspec) ...] [if] [in], saving(filespec) [simopts]
```

ysimspec is `{_ysim#}` or `{_ysim#[`*numlist*`]}`, where `{_ysim#}` refers to all observations of the #th simulated outcome and `{_ysim#[`*numlist*`]}` refers to the selected observations, *numlist*, of the #th simulated outcome. `{_ysim}` is a synonym for `{_ysim1}`. With large datasets, specification `{_ysim#}` may use a lot of time and memory and should be avoided. See *Generating and saving simulated outcomes*.

funcspec is one of the following,

```
[label:]@func(arg1 [, arg2])
[label:]@userprog arg1 [arg2] [, extravars(varlist) passthruopts(string)]
```

where *label* is a valid Stata name; *func* is an official or user-defined Mata function that operates on column vectors and returns a real scalar; *userprog* is a user-defined Stata program; and *arg1* and *arg2* are one of `{_ysim[#]}`, `{_resid[#]}`, or `{_mu[#]}`. `{_mu#}` refers to expected values, and `{_resid#}` refers to residuals for the #th outcome, where the latter is defined as the difference between `{_ysim#}` and `{_mu#}`. *arg2* is primarily for use with user-defined Mata functions; see *Defining test statistics using Mata functions*.

Compute posterior summaries of simulated outcomes

Posterior mean of simulated outcomes

```
bayespredict [type] newvarspec [if] [in], mean
    [outcome(depvar) meanopts simopts]
```

Posterior median or posterior standard deviation of simulated outcomes

```
bayespredict [type] newvarspec [if] [in], median|std
    [outcome(depvar) simopts]
```

Credible intervals for simulated outcomes

```
bayespredict [type] newvar_l newvar_u [if] [in], cri
    [outcome(depvar) criopts simopts]
```

newvarspec is *newvar* for single-outcome models and *newvarlist* or *stub*`*` for multiple-outcome models.

Generate a subset of MCMC replicates of simulated outcomes

`bayesreps` [*type*] *newrepspec* [*if*] [*in*], `nreps(#)` [`outcome(`*depvar*`)` *simopts*]

newrepspec is *newvar* with `nreps(1)` for a single replicate and *stub*`*` with `nreps(#)`, where # is greater than 1, for multiple replicates.

meanopts	Description
Main	
`mcse(`*newvar*`)`	create *newvar* containing MCSEs
Advanced	
`batch(#)`	specify length of block for batch-means calculations; default is `batch(0)`
`corrlag(#)`	specify maximum autocorrelation lag; default varies
`corrtol(#)`	specify autocorrelation tolerance; default is `corrtol(0.01)`

simopts	Description
Simulation	
`rseed(#)`	random-number seed
* `chains(_all` | *numlist*`)`	specify which chains to use for computation; default is `chains(_all)`
`dots`	display dots every 100 iterations and iteration numbers every 1,000 iterations
`dots(#[, every(#)])`	display dots as simulation is performed

*Option `chains()` is relevant only when option `nchains()` is used with `bayesmh`.

criopts	Description
Main	
`clevel(#)`	set credible interval level; default is `clevel(95)`
`hpd`	calculate HPD credible intervals instead of the default equal-tailed credible intervals

Options

Options are presented under the following headings:

Options for predictions
Options for posterior summaries
Options for bayesreps

Options for predictions

Main

`saving(`*filename*`[ , replace ])` saves the requested predictions such as simulated outcomes and residuals in *filename*`.dta`. It also saves auxiliary estimation results in *filename*`.ster`, which is accessible by specifying `estimates use` *filename*. The `replace` option specifies to overwrite *filename*`.dta` and *filename*`.ster` if they exist. `saving()` is required when computing predictions. The results are saved only for the outcome variables, observations, and functions that are specified with `bayespredict`. See *Prediction dataset* for details.

`extravars(`*varlist*`)` is for use with user-defined Stata programs. It specifies any variables in addition to dependent and independent variables that you may need to calculate predictions. For example, such variables are offset variables and exposure variables for count-data models.

`passthruopts(`*string*`)` is for use with user-defined Stata programs. It specifies a list of options you may want to pass to your program when calculating predictions. For example, these options may contain fixed values of model parameters and hyperparameters.

Simulation

`rseed(#)` sets the random-number seed. This option can be used to reproduce results. With one chain, `rseed(#)` is equivalent to typing `set seed #` prior to calling `bayespredict`; see [R] **set seed**. With multiple chains, you should use `rseed()` for reproducibility; see *Reproducing results* in [BAYES] **bayesmh**.

`chains(_all | `*numlist*`)` specifies which chains from the MCMC sample to use for computation. The default is `chains(_all)` or to use all simulated chains. Using multiple chains, provided the chains have converged, generally improves MCMC summary statistics. Option `chains()` is relevant only when option `nchains()` is specified with `bayesmh`.

`dots` and `dots(#)` specify to display dots during simulation. With multiple chains, these options affect all chains. `dots(#)` displays a dot every # iterations. If `dots(..., every(#))` is specified, then an iteration number is displayed every #th iteration instead of a dot. `dots(, every(#))` is equivalent to `dots(1, every(#))`. `dots` displays dots every 100 iterations and iteration numbers every 1,000 iterations; it is a synonym for `dots(100, every(1000))`.

Options for posterior summaries

Main

`mean` calculates posterior means of a simulated outcome variable and stores them as a new variable in the current dataset.

`median` calculates posterior medians of a simulated outcome variable and stores them as a new variable in the current dataset.

`std` calculates posterior standard deviations of a simulated outcome variable and stores them as a new variable in the current dataset.

`mean`, `median`, and `std` can compute results for all simulated outcome variables or for a specific one. To compute results for all simulated outcome variables, you specify p new variables, where p is the number of dependent variables. Alternatively, you can specify *stub*`*`, in which case these options will store the results in variables *stub*`1`, *stub*`2`, ..., *stubp*. To compute the results for a specific simulated outcome variable, you specify one new variable and, optionally, the outcome variable name in option `outcome()`; if you omit `outcome()`, the first outcome variable is assumed.

`cri` calculates credible intervals for a simulated outcome variable and stores the corresponding lower and upper bounds in two new variables in the current dataset. For multiple-outcome models, it computes the results for the outcome variable as specified in option `outcome()` or, by default, for the first outcome variable.

`outcome(`*depvar*`)` is for use with multiple-outcome models when computing posterior summaries of simulated outcomes. It specifies for which simulated outcome posterior summaries are to be calculated. `outcome()` should contain a name of the outcome (dependent) variable. The default is the first outcome variable. `outcome()` may not be combined with the *newvarlist* or *stub*`*` specification.

`mcse(`*newvar*`)` is for use in a combination with option `mean`. It adds *newvar* of storage type *type* containing MCSEs for the posterior means of a simulated outcome variable. If multiple variables are specified with `bayespredict`, *newvar* is used as a stub *newvar*`*`.

`clevel(#)` specifies the credible level, as a percentage, for equal-tailed and HPD credible intervals. The default is `clevel(95)` or as set by [BAYES] **set clevel**. This option requires that `cri` also be specified.

`hpd` calculates the HPD credible intervals instead of the default equal-tailed credible intervals. This option requires that `cri` also be specified.

Simulation

`rseed(#)` sets the random-number seed. This option can be used to reproduce results. With one chain, `rseed(#)` is equivalent to typing `set seed #` prior to calling `bayespredict`; see [R] **set seed**. With multiple chains, you should use `rseed()` for reproducibility; see *Reproducing results* in [BAYES] **bayesmh**.

`chains(_all` | *numlist*`)` specifies which chains from the MCMC sample to use for computation. The default is `chains(_all)` or to use all simulated chains. Using multiple chains, provided the chains have converged, generally improves MCMC summary statistics. Option `chains()` is relevant only when option `nchains()` is specified with `bayesmh`.

`dots` and `dots(#)` specify to display dots during simulation. With multiple chains, these options affect all chains. `dots(#)` displays a dot every # iterations. If `dots(..., every(#))` is specified, then an iteration number is displayed every #th iteration instead of a dot. `dots(, every(#))` is equivalent to `dots(1, every(#))`. `dots` displays dots every 100 iterations and iteration numbers every 1,000 iterations; it is a synonym for `dots(100, every(1000))`.

Advanced

The advanced options are available only in a combination with option `mean`.

`batch(#)` specifies the length of the block for calculating batch means and MCSE using batch means. The default is `batch(0)`, which means no batch calculations. When `batch()` is not specified, MCSE is computed using effective sample sizes instead of batch means. Option `batch()` may not be combined with `corrlag()` or `corrtol()`.

`corrlag(#)` specifies the maximum autocorrelation lag used for calculating effective sample sizes. The default is $\min\{500, \texttt{mcmcsize()}/2\}$. The total autocorrelation is computed as the sum of all lag-k autocorrelation values for k from 0 to either `corrlag()` or the index at which the autocorrelation becomes less than `corrtol()` if the latter is less than `corrlag()`. Options `corrlag()` and `batch()` may not be combined.

`corrtol(#)` specifies the autocorrelation tolerance used for calculating effective sample sizes. The default is `corrtol(0.01)`. For a given model parameter, if the absolute value of the lag-k

autocorrelation is less than corrtol(), then all autocorrelation lags beyond the kth lag are discarded. Options corrtol() and batch() may not be combined.

Options for bayesreps

Main

nreps(#) specifies the number of MCMC replicates of simulated outcomes to be drawn at random from the entire sample of MCMC replicates. # must be an integer between 1 and the MCMC sample size, inclusively. The generated replicates are stored as new variables in the current dataset. For a single replicate, nreps(1), you specify one new variable name. For multiple replicates, you specify a *stub*∗, in which case the replicates will be stored in variables *stub*1, *stub*2, ..., *stub*R, where R is the number of replicates specified in nreps().

outcome(*depvar*) is for use with multiple-outcomes models when generating MCMC replicates of simulated outcomes using bayesreps. It specifies for which simulated outcome MCMC replicates are to be generated. The default is to use the first outcome variable. You can specify other outcome (dependent) variable names in outcome().

Simulation

rseed(#) sets the random-number seed. This option can be used to reproduce results. With one chain, rseed(#) is equivalent to typing set seed # prior to calling bayespredict; see [R] **set seed**. With multiple chains, you should use rseed() for reproducibility; see *Reproducing results* in [BAYES] **bayesmh**.

chains(_all | *numlist*) specifies which chains from the MCMC sample to use for computation. The default is chains(_all) or to use all simulated chains. Using multiple chains, provided the chains have converged, generally improves MCMC summary statistics. Option chains() is relevant only when option nchains() is specified with bayesmh.

dots and dots(#) specify to display dots during simulation. With multiple chains, these options affect all chains. dots(#) displays a dot every # iterations. If dots(..., every(#)) is specified, then an iteration number is displayed every #th iteration instead of a dot. dots(, every(#)) is equivalent to dots(1, every(#)). dots displays dots every 100 iterations and iteration numbers every 1,000 iterations; it is a synonym for dots(100, every(1000)).

Remarks and examples

stata.com

Remarks are presented under the following headings:

Examples are presented under the following headings:

Overview of Bayesian predictions

Bayesian analysis rests on the assumptions that model parameters are random quantities distributed according to some prior beliefs and that the data, once observed, are fixed. The main goal of Bayesian inference is to estimate the posterior distribution of model parameters, which combines the prior beliefs with evidence from the observed data, and form inferences about these parameters. But what if we want to estimate a future outcome value? This is one of the goals of Bayesian prediction.

Bayesian predictions are useful in a wide range of applications. They can be used as optimal predictors in forecasting, optimal classifiers in classification problems, imputations for missing data, and more. They are also important for checking model goodness of fit.

Bayesian prediction differs from frequentist prediction. Prediction, in a frequentist sense, is a deterministic function of estimated model parameters. For example, in a linear regression, the linear predictor, which is a linear combination of estimated regression coefficients and observed covariates, is used to predict values of continuous outcomes. Bayesian predictions, on the other hand, are functions of simulated outcomes and are thus stochastic quantities. Simulated outcomes are new outcome values generated from the so-called posterior predictive distribution, which we describe next.

Prior and posterior predictive distributions

Before the data $\mathbf{y}$ are observed, the distribution of $\mathbf{y}$ is

$$p(\mathbf{y}) = \int p(\mathbf{y}, \boldsymbol{\theta}) d\boldsymbol{\theta} = \int p(\mathbf{y}|\boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} \tag{1}$$

where $p(\mathbf{y}|\boldsymbol{\theta})$ is the likelihood of $\mathbf{y}$ given model parameters $\boldsymbol{\theta}$ and $p(\boldsymbol{\theta})$ is the prior distribution for $\boldsymbol{\theta}$. $p(\mathbf{y})$ is the so-called prior predictive distribution, which is more commonly known as the marginal distribution of $\mathbf{y}$.

Suppose that $\mathbf{y}^{\text{obs}}$ are observed data and $\mathbf{y} = \mathbf{y}^{\text{new}}$ are new, unobserved (future) data. The posterior predictive distribution of $\mathbf{y}^{\text{new}}$ is

$$p(\mathbf{y}^{\text{new}}|\mathbf{y}^{\text{obs}}) = \int p(\mathbf{y}^{\text{new}}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{y}^{\text{obs}}) d\boldsymbol{\theta} \tag{2}$$

where $p(\boldsymbol{\theta}|\mathbf{y}^{\text{obs}})$ is the posterior distribution of $\boldsymbol{\theta}$. You can think of a posterior predictive distribution (2) as a prior predictive distribution (1) updated after observing the data $\mathbf{y}^{\text{obs}}$.

Simulated outcomes

Like the posterior distribution of model parameters, the predictive distribution $p(\mathbf{y}^{\text{new}}|\mathbf{y}^{\text{obs}})$ usually does not have a closed form and must be approximated. The goal of Bayesian prediction is to simulate data from $p(\mathbf{y}^{\text{new}}|\mathbf{y}^{\text{obs}})$. We will refer to these data as *simulated outcomes*, $\mathbf{y}^{\text{sim}}$.

Formula (2) provides a way of simulating new outcome values by using a two-step procedure. First, model parameters $\boldsymbol{\theta}^{\star}$ are simulated from their posterior distribution $p(\boldsymbol{\theta}|\mathbf{y}^{\text{obs}})$. Then, the new outcome values $\mathbf{y}^{\text{sim}}$ are simulated from the likelihood model $p(\mathbf{y}^{\text{sim}}|\boldsymbol{\theta}^{\star})$ using the simulated model parameters from step 1. These two steps are repeated for a prespecified number of MCMC iterations, T. The result is an MCMC sample of simulated outcomes, $(\mathbf{y}^{\text{sim},1}, \mathbf{y}^{\text{sim},2}, \ldots, \mathbf{y}^{\text{sim,T}})$. This sample is used to estimate the posterior predictive distribution.

Thus, unlike classical prediction, which produces a single value for each observation, Bayesian prediction produces a sample of T simulated values for each observation. If you have n observations in the dataset, the result of a Bayesian prediction will be a $T \times n$ matrix (for each outcome or dependent variable). Therefore, Bayesian predictions are often computed for a subset of observations or for various summaries over observations such as means, quantiles, minimum and maximum values, and so on. Sometimes, a smaller sample of $R << T$ MCMC replicates of simulated outcomes is used to explore the posterior distribution of simulated outcomes. In other cases, posterior summaries over the MCMC replicates such as posterior means and medians of simulated outcomes may be of interest.

Posterior predictive checking and replicated outcomes

In addition to predicting future observations, Bayesian prediction is useful for model checking. Model checking is accomplished by performing the so-called posterior predictive checks, which compare various characteristics of the posterior predictive distribution with those observed in the data.

The concept of replicated data or replicated outcomes arises in the context of posterior predictive checking for regression-type models. In a regression setting, the posterior predictive distribution also depends on the covariate-data matrix X, $p(\mathbf{y}^{\rm new}|\mathbf{y}^{\rm obs}) = p(\mathbf{y}^{\rm new}|\mathbf{y}^{\rm obs}, X)$. The data matrix X may contain the observed values that were used to fit the Bayesian model, $X^{\rm obs}$, or the new values, $X^{\rm new}$. *Replicated outcomes* are outcomes simulated from the posterior predictive distribution, $p(\mathbf{y}^{\rm new}|\mathbf{y}^{\rm obs}, X^{\rm obs})$, using the observed covariate data. In other words, the replicated outcomes are the outcomes we would observe if we repeated our experiment again. We will denote replicated outcomes as $\mathbf{y}^{\rm rep}$.

Replicated outcomes are also known as in-sample predictions, whereas outcomes simulated using new covariate data, $X^{\rm new}$, are known as out-of-sample predictions. In-sample predictions are useful for diagnostic checks. Out-of-sample predictions can be used for forecasting and model validation. In the latter case, the data are split into training and test subsamples: the training subsample is used to fit a Bayesian model, and the test subsample is used to assess prediction accuracy of the fitted model.

Posterior predictive checking is performed by comparing the distribution (or certain aspects of it) of the replicated data to that of the observed data. This can be done visually by examining histograms and quantile plots. More formally, discrepancy measures such as a mean, minimum, and maximum statistics computed for the replicated data and for the observed data can be compared using posterior predictive p-values; see [BAYES] **bayesstats ppvalues** for details.

It is important to realize the difference between MCMC diagnostic checks (*Convergence of MCMC* in [BAYES] **bayesmh**) and posterior predictive checks. The former examines the properties of MCMC sampling, whereas the latter inspects how well the specified Bayesian model describes the observed data. But these two types of checks are related—an ill-fitting model lowers the MCMC sampling efficiency and may even lead to nonconvergence of the MCMC algorithm.

For in-depth coverage of Bayesian predictions and posterior predictive inference, see Meng (1994), West (1986), Tsui and Weerahandi (1989), Gelman, Meng, and Stern (1996), Gelman and Rubin (1992), and Gelman et al. (2014), to name a few.

Using bayespredict and bayesreps

`bayespredict` computes Bayesian predictions using current estimation results produced by the `bayesmh` command with built-in likelihood models and saves them in a separate Stata dataset. Bayesian predictions include simulated outcomes, which are samples from the posterior predictive distribution of the fitted Bayesian model, and their functions. You can also compute posterior summaries of simulated outcomes and store them as new variables in the current dataset.

To compute Bayesian predictions, you must specify the `saving()` option with `bayespredict` to save the prediction results; see *Generating and saving simulated outcomes*. To compute posterior summaries, you must specify one or more new variable names and the corresponding option such as `mean` for posterior mean and `std` for posterior standard deviation; see *Posterior summaries of simulated outcomes*.

`bayesreps` generates a random subset of MCMC replicates of simulated outcomes from the entire MCMC sample and stores them as new variables in the current dataset. This command is useful for checking model fit. The number of replicates is specified in the `nreps(`$\#_{\text{reps}}$`)` option. With multiple replicates, you must specify a variable *stub*`*` with `bayesreps`, and the command will generate new variables *stub*1, *stub*2, ..., *stub*$\#_{\text{reps}}$ in the current dataset. For multiple-outcome models, the replicates are produced for one outcome at a time. The first outcome is the default, but you can specify a different outcome variable in the `outcome()` option.

Both `bayespredict` and `bayesreps` require that `bayesmh`'s MCMC simulation dataset be saved prior to their execution. You can save MCMC simulation results by specifying the `saving()` option with `bayesmh` during or after estimation; see *Storing estimation results after Bayesian estimation* in [BAYES] **Bayesian postestimation**.

Both commands produce stochastic results. Use the `rseed()` option for reproducibility. Depending on the number of observations, the specified MCMC sample size, and model complexity, the computations may be time consuming. Options `dots` and `dots()` may be useful in this case to monitor the progress. They display a dot for each simulation performed.

`bayespredict` and `bayesreps` can be used to make in-sample or out-of-sample predictions; see *Description* in [R] **predict** for how to specify such predictions.

Generating and saving simulated outcomes

Generating and saving simulated outcomes is the main usage of `bayespredict`, which requires the `saving()` option when generating simulated outcomes. The simplest specification is

```
. bayespredict {_ysim1}, saving(filename)
```

which generates the simulated values for the first outcome variable and saves them in *filename*`.dta`. You can also use `{_ysim}` as a synonym for `{_ysim1}`.

The above specification produces the prediction dataset *filename*`.dta`, which contains T observations and n variables, where T is the MCMC sample size used by `bayesmh` and n is the number of observations in the original dataset. That is, an MCMC sample of size T is generated for each observation of the outcome variable.

For example, if our dataset has 100 observations and we use an MCMC sample of size 10,000 during simulation, `bayespredict` will produce the prediction dataset *filename*`.dta` with 10,000 observations and 100 variables. This specification may not always be feasible, especially for large datasets, or even necessary.

You would rarely need to simulate and store all observations for all outcome variables. More likely, if you are performing model diagnostics, you may be interested only in several test statistics, which you can simulate without storing the simulated outcomes; see *Defining test statistics using Mata functions*. Or you may be interested only in posterior summaries of simulated outcomes; see *Posterior summaries of simulated outcomes*. Or you may need to explore only a small random subset of MCMC replicates of simulated outcomes, which you can obtain by using the `bayesreps` command. Or if you are interested in forecasting, you may need to simulate values for only a few new data points.

For example, suppose we want to simulate outcome values for 10 new observations only, which are stored in observations 101 through 110 in our original dataset. We can do this using

```
. bayespredict {_ysim1[101/110]}, saving(filename)
```

or, equivalently, using

```
. bayespredict {_ysim1} in 101/110, saving(filename)
```

The two specifications above are more efficient with respect to execution time and storage.

The full syntax of `bayespredict` for simulating all variables and all observations is

```
. bayespredict {_ysim1} {_ysim2} ..., saving(filename)
```

where you specify `{_ysim#}` for the #th outcome variable. The order of variables is determined by the order in which they were specified with `bayesmh`.

If you need to predict multiple outcomes, it may be more efficient with regard to storage to simulate them separately. Remember that the total number of variables in the prediction dataset may not exceed the current `c(maxvar)` setting. Because `bayespredict` stores additional variables, the number of specified outcome observations may not exceed `floor((c(maxvar)-3)/2)`; see *Prediction dataset*.

By default, `bayespredict` computes out-of-sample predictions. This may sometimes lead to missing predicted observations, for instance, when some of the covariates contain missing values. In the context of `bayespredict` when simulating outcomes, residuals, and expected values, this implies that the prediction dataset may contain variables containing all missing observations. Recall that the variables in the prediction dataset correspond to the observations in the original dataset. In such cases, to reduce the size of the prediction dataset, you may consider restricting the prediction sample to the estimation sample, `if e(sample)`; or specifying a subset of observations using *numlist*, for example, `_ysim[`*numlist*`]`; or specifying the subset of interest by using `if` and `in`.

Defining test statistics using Mata functions

Instead of simulating all observations for your outcomes of interest, you may be interested in obtaining only some summary statistics such as sample means, medians, smallest and largest observations, and standard deviations calculated over these observations. This is commonly used when performing posterior predictive checks; see *Posterior predictive inference*.

Test statistics are scalar functions of observed (or simulated) outcome values. Let `y` be an outcome variable in a dataset of size n and let $\mathbf{y}^{\text{sim}} = (y_1^{\text{sim}}, y_2^{\text{sim}}, \ldots, y_n^{\text{sim}})^T$ denote one simulated outcome sample given as a column vector. A test statistic $T(\mathbf{y}^{\text{sim}})$ summarizes the column vector $\mathbf{y}^{\text{sim}}$ by a single number. For example, the mean statistic is defined as

$$T(\mathbf{y}^{\text{sim}}) = \frac{1}{n}(y_1^{\text{sim}} + \cdots + y_n^{\text{sim}}) = \overline{\mathbf{y}}^{\text{sim}}$$

In `bayespredict`, test statistics can be defined using Mata functions or Stata programs. Here we focus on the specifications using Mata functions; see *User-defined Stata programs* for Stata programs. Note that if you need to compute a test quantity, $T(\mathbf{y}, \boldsymbol{\theta})$, that directly uses model parameters $\boldsymbol{\theta}$, you must use Stata programs.

`bayespredict` supports Mata functions that return a scalar and accept one or two column vectors as arguments. You can specify the following as the arguments to the Mata functions: simulated outcomes, `{_ysim#}`; simulated residuals, `{_resid#}`; and expected outcome values, `{_mu#}`. `{_resid#}` is defined as the difference between `{_ysim#}` and `{_mu#}`. (Specifications `{_resid#}` and `{_mu#}` are not available for ordinal models.) You can also use `{_ysim}`, `{_resid}`, and `{_mu}` as synonyms

for {_ysim1}, {_resid1}, and {_mu1}, respectively. If you used if or in with bayespredict to restrict the prediction sample or specified only a subset of observations, that is, {_ysim[1/10]}, the column vectors passed to Mata functions as arguments will contain only the available observations.

Suppose we want to produce an MCMC sample of means of the first simulated outcome. We can specify

```
. bayespredict (@mean({_ysim1}), saving(...)
```

Similarly, we can produce an MCMC sample of means for the residuals of the first simulated outcome

```
. bayespredict (resmean: @mean({_resid1}), saving(...)
```

In the above, we also labeled our prediction as resmean. We can use this label to refer to this prediction in other Bayesian postestimation commands such as bayesstats ppvalues and bayesstats summary. If we do not specify our own labels, the default labels will be used for each prediction. The default label is *arg1_func*, where *arg1* is the first function argument and *func* is the name of the function. For instance, in our first example, the default label _ysim1_mean will be used.

You will typically specify only one argument with most official Mata functions. The support of two arguments is provided primarily for calculating more complicated test statistics using user-defined Mata functions. For example, let's define a new Mata function that calculates the sum of squared Pearson residuals assuming a Poisson model.

We define a Mata function, sumpresid(), that calculates the squared Pearson residuals as the squared difference between the simulated outcome vector, ysim, and expected values, mu, divided by the variance, which is also mu for a Poisson model. The result is the sum of these squared standardized differences.

```
mata:
    real scalar sumpresid(real colvector ysim, real colvector mu) {
        return (sum((ysim-mu):^2:/mu))
    }
end
```

Then, we can call bayespredict with the following specification to compute the sum of squared Pearson residuals for the first outcome in the model:

```
. bayespredict (@sumpresid({_ysim1}, {_mu1})), saving(...)
```

Mata functions can be used only with one outcome at a time. That is, specifications that refer to two outcomes such as @myprog({_ysim1}, {_ysim2}), @myprog({_ysim1}, {_mu2}), or @myprog({_ysim1}, {_resid2}) are not allowed.

Mata functions are preferable to Stata programs because of speed, but Stata programs provide more flexibility to compute complicated functions; see *User-defined Stata programs* below.

User-defined Stata programs

Mata functions (see *Defining test statistics using Mata functions*) are more efficient and faster in computing simple test statistics and test quantities, but they have limitations. For example, you cannot access model parameters within Mata functions. You can within Stata programs. Although executing Stata programs may be much slower, they provide more flexibility for computing test quantities.

A Stata program must have the following format in order to be used by bayespredict:

```
program userprog
        version 17.0
        args res simvar1 [simvar2]
        ... computation ...
        scalar 'res' = ...
end
```

The first argument, *res*, contains the name of a temporary scalar to store the final result. The second argument, *simvar1*, and the third (optional) argument, *simvar2*, contain the names of temporary variables, which store the simulation results for the quantities specified as program arguments *arg1* and *arg2* with bayespredict:

```
. bayespredict ([label]: @userprog arg1 [arg2]), saving(...) ...
```

arg1 and *arg2* may be one of {_ysim#}, {_mu#}, or {_resid#}, but they should refer to the same outcome variable; that is, they must use the same *#*. *label* is the label for the computed prediction result that can be used later to refer to this result within other Bayesian postestimation commands such as bayesstats summary. If we do not specify our own label, the default label will be used for each prediction. The default label is *arg1*_*userprog*, where *arg1* is the first program argument and *userprog* is the name of the program.

Recall the sumpresid() Mata function defined in the previous section. Below, we replicate the same computation but now using the Stata program.

```
program sumpresidprog
        version 17.0
        args sum ysim mu
        tempvar presid
        generate double 'presid' = ('ysim'-'mu')^2/'mu'
        summarize 'presid', meanonly
        scalar 'sum' = r(sum)
end
```

We can then call bayespredict with the following specification,

```
. bayespredict (@sumpresidprog {_ysim1} {_mu1}), saving(...)
```

to compute this statistic for the first outcome. Because we did not specify our own label in the above, the default label _ysim1_sumpresidprog will be used.

Generally, our Stata program should use a proper "touse" variable, which marks the prediction sample of bayespredict. Unlike Mata functions, the prediction results passed to Stata programs as arguments will contain all observations. However, the observations outside the prediction sample will contain missing values. Nevertheless, it is good practice to always use the touse variable in the calculations.

```
program sumpresidprog
        version 17.0
        args sum ysim mu
        local touse $BAYESPR_touse
        tempvar presid
        generate double 'presid' = ('ysim'-'mu')^2/'mu' if 'touse'
        summarize 'presid' if 'touse', meanonly
        scalar 'sum' = r(sum)
end
```

The global macro $BAYESPR_touse contains a temporary name of a binary variable that marks the prediction sample, which we now use in our calculations.

One flexibility of Stata programs is that we can access model parameters within them. In the above programs, we used precomputed expected values, `mu`. We can compute these values manually by using the simulated model parameters and observed variables.

```
program sumpresidprogmu
        version 17.0
        args sum ysim
        local touse $BAYESPR_touse
        local theta $BAYESPR_theta                              //<--New line
        tempvar xb mu                                           //<--New line
        matrix score double `xb' = `theta' if `touse'           //<--New line
        qui generate double `mu' = invlogit(`xb') if `touse' //<--New line
        tempvar presid
        generate double `presid' = (`ysim'-`mu')^2/`mu' if `touse'
        summarize `presid' if `touse', meanonly
        scalar `sum' = r(sum)
end
```

To compute expected values, we need to compute the linear predictor. To compute the linear predictor, we need coefficient estimates. The coefficient estimates are provided in a temporary matrix (row vector) with the name stored in the global macro `$BAYESPR_theta`. The columns of this temporary matrix are labeled properly with the names of the corresponding predictors, so we can use `matrix score` (see [P] **matrix score**) to easily compute the linear predictor. We then use the inverse-logit function to compute expected values (probabilities) from the linear predictions. The rest of the program is the same as earlier.

We call the above program using the following `bayespredict` specification:

```
. bayespredict (@sumpresidprogmu {_ysim1}), saving(...)
```

See example 8.

For some programs, you may need to pass additional variables or contents of command options. You can use `extravars()` and `passthruopts()` for that; see *Options for predictions*.

You can access the following global macros from the Stata programs used with `bayespredict`.

Global macros	Description
`$BAYESPR_theta`	name of a temporary matrix (row vector) of scalar parameters; stripes are properly named after the names of model parameters
`$BAYESPR_matrix_`*mname*	name of a temporary matrix containing simulated matrix parameter *mname*
`$BAYESPR_touse`	variable containing 1 for the observations to be used; 0 otherwise
`$BAYESPR_extravars`	*varlist* specified in `extravars()`
`$BAYESPR_passthruopts`	options specified in `passthruopts()`

Posterior summaries of simulated outcomes

In some applications, we may not need the actual simulated outcomes but rather their posterior summaries such as posterior means, medians, and standard deviations. For this purpose, `bayespredict` offers the `mean`, `median`, `std`, and `cri` options to compute posterior means, medians, standard deviations, and credible intervals. When you specify these options, the prediction results are stored in the specified new variables in the current dataset. You do not need to specify the `saving()` option in this case because the high-dimensional simulation outcomes are not saved, only their posterior summaries.

With mean, median, and std, you can compute results for one outcome variable at a time or for all outcome variables. In the first case, you specify a new variable name and the name of the outcome (dependent) variable in the outcome() option. If you omit outcome(), the first outcome variable will be used. To compute results for all outcome variables, you specify a new variable name for each outcome or *stub**, in which case the new variables will be named *stub*1, *stub*2, and so on.

When you compute posterior means, you can also specify the mcse(*newvar*) option to compute their corresponding MCSEs. If posterior means are computed for multiple outcome variables, *newvar* is used as *stub** to store MCSEs for each outcome in *newvar*1, *newvar*2, and so on.

With cri, you specify two new variable names to contain the lower and upper credible bounds. You can compute results only for one outcome variable at a time, which you specify in the outcome() option. If you omit this option, the first outcome variable is assumed. You can specify the clevel() option to change the default 95% credible level and the hpd option to calculate HPD credible intervals instead of the default equal-tailed intervals.

All computed results are stochastic. You should specify the rseed() option for reproducibility. Also see *Syntax* for other available simulation options, *simopts*.

Prediction dataset

bayespredict saves prediction results in a dataset *filename*.dta as specified in the saving(*filename*) option. In addition, bayespredict stores auxiliary estimation results, described in *Stored results*, in *filename*.ster. This file is used by other postestimation commands such as bayesstats summary when summarizing the simulated prediction quantities.

The format of the *filename*.dta file is similar to the simulation dataset created by the bayesmh command. The first two variables are _chain and _index, which store the respective chain and MCMC iteration identifiers. Following are the variables containing simulated values for the $\#_1$th outcome variable and the $\#_2$th observation, _ysim$\#_1$_$\#_2$, if any, and the corresponding expected outcome values, _mu$\#_1$_$\#_2$. For any function of simulated outcomes or residuals specified with bayespredict, there are two variables in the dataset named *label* and _obs_*label*, where *label* is the specified function or program label. Variable *label* contains the MCMC sample of values of the function. Variable _obs_*label* contains the observed values of the function, which are computed by substituting the simulated outcome for the observed outcome variable in the function specification. This variable is consumed by [BAYES] **bayesstats ppvalues**. Finally, the _frequency variable is the last variable in the prediction dataset. It always contains one in the prediction dataset and is provided purely for the consistency with the simulation dataset, where it records the frequency of duplicate sets of model parameters.

If bayespredict is specified with p simulated outcomes, each with n observations, and with k functions or programs, then the prediction dataset will contain $2pn + 2k + 3$ variables. The number of observations in the prediction dataset is determined by the MCMC sample size, T, used by bayesmh.

After your analysis, if you no longer need the prediction dataset, remember to remove both *filename*.dta and *filename*.ster.

Bayesian predictions

Consider the rare infectious disease example from Hoff (2009) that we analyzed in *Beta-binomial model* of [BAYES] **bayesmh**. A small random sample of 20 subjects from a city is checked for infection, and none is observed to be infected. The parameter of interest θ, $\theta \in [0, 1]$, is the proportion of infected individuals in the city. The outcome y is the number of infected subjects in the sample of 20. The sampling distribution for the outcome y is thus assumed to be binomial, $y|\theta \sim \text{binomial}(20, \theta)$.

Our observed data contain one observation that is zero because we did not observe any infected subjects in our sample. We can easily generate these data as follows:

```
. set obs 1
Number of observations (_N) was 0, now 1.
. generate byte y = 0
```

Following the examples in *Beta-binomial model* (except we are using a different random-number seed here), we assume a beta$(2, 20)$ prior for θ and use `bayesmh` to fit the resulting beta-binomial model.

```
. bayesmh y, likelihood(dbinomial({theta}, 20))
> prior({theta}, beta(2, 20)) saving(betabin_mcmc) rseed(16)
Burn-in ...
Simulation ...
Model summary
------------------------------------------------------------------------------
Likelihood:
  y ~ binomial({theta},20)
Prior:
  {theta} ~ beta(2,20)
------------------------------------------------------------------------------

Bayesian binomial model                         MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling        Burn-in          =      2,500
                                                MCMC sample size =     10,000
                                                Number of obs    =          1
                                                Acceptance rate  =      .4627
Log marginal-likelihood = -1.1575104            Efficiency       =      .1446

------------------------------------------------------------------------------
             |                                                 Equal-tailed
             |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
       theta |  .0476128   .0320509   .000843   .0406464   .0057875   .1251631
------------------------------------------------------------------------------
file betabin_mcmc.dta saved.
```

The posterior mean for `{theta}`, which is also the probability that a subject from a sample of 20 will be infected, is estimated to be 0.0476. Thus, we would expect $20 \times 0.0476 = 0.952$ infected subjects in a sample of 20.

Let's explore various Bayesian predictions for this beta-binomial model. The relevant examples are presented under the following headings:

▷ Example 1: Predicting the number of infected subjects

Let's predict the number of infected subjects, our outcome, assuming the fitted beta-binomial model. To do this in a Bayesian framework, we need to simulate the outcome from its posterior predictive distribution. We can use `bayespredict` to do this.

To use `bayespredict`, we must first save our MCMC simulation results from `bayesmh` in a dataset, which we already did by specifying the `saving(betabin_mcmc)` option with `bayesmh`. If you forget to specify this option during estimation, you can always do it after by typing

```
. bayesmh, saving(betabin_mcmc)
```

We simulate the outcome by specifying {_ysim} with bayespredict and save the simulated data in betabin_pred.dta; the saving() option is required with bayespredict when simulating Bayesian predictions. Because the command uses simulation, we also specify the rseed() option for reproducibility.

```
. bayespredict {_ysim}, saving(betabin_pred) rseed(16)
Computing predictions ...
file betabin_pred.dta saved.
file betabin_pred.ster saved.
```

The computation may be time consuming, so the command displays Computing predictions ... to inform you that the computation is in progress. You may also specify the dots or dots() option to see the dots as simulations are performed.

In addition to saving prediction results in a Stata dataset, bayespredict also saves auxiliary estimation results in the betabin_pred.ster file. This file is used by other postestimation commands such as bayesstats summary when summarizing the simulated prediction quantities. Remember to remove this file in addition to your prediction dataset when you no longer need them.

The bayespredict command simulates T outcome values for each specified outcome and for each specified observation. T is the MCMC sample size used by bayesmh. The outcome values are simulated for each set of T MCMC estimates of model parameters generated by bayesmh. Our bayespredict specification {_ysim} is equivalent to {_ysim1} and refers to all observations of the first outcome. In our example, we have only one observation and one outcome, and the default MCMC sample size is 10,000. Thus, betabin_pred.dta contains one simulated variable, _ysim1_1, and 10,000 observations, in addition to other auxiliary variables such as chain and iteration number identifiers; see *Prediction dataset*.

```
. describe using betabin_pred
Contains data
 Observations:        10,000                  2 Jun 2021 18:54
    Variables:             5
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
_chain          int     %8.0g                 Chain identifier
_index          int     %8.0g                 Iteration number
_ysim1_1        double  %10.0g                Simulated y, obs #1
_mu1_1          double  %10.0g                Expected values for y, obs #1
_frequency      byte    %8.0g                 Frequency weight
-------------------------------------------------------------------------------
Sorted by:
```

In this dataset, _ysim1_1 represents an MCMC sample of size 10,000 from the posterior predictive distribution of y for the first observation. If we had more observations, say, 100, the dataset would have contained 100 variables, _ysim1_1, _ysim1_2, ..., _ysim1_100, one for each observation. In the prediction dataset, the observations are MCMC replicates, and the variables are outcome values for each observation and each outcome from the data that were used to fit the model.

◁

Example 2: Summarizing prediction results

We can summarize our prediction results like any other Bayesian model parameter. For example, we can calculate standard posterior summaries for {_ysim} by using bayesstats summary.

```
. bayesstats summary {_ysim} using betabin_pred
Posterior summary statistics                       MCMC sample size =    10,000
```

	Mean	Std. dev.	MCSE	Median	Equal-tailed [95% cred. interval]	
_ysim1_1	.9526	1.145899	.020218	1	0	4

The calculated posterior predictive mean is 0.95, which agrees with our earlier computation of $20 \times 0.0476 = 0.952$ using the posterior mean estimate of θ, 0.0476. Under our Bayesian model, we should expect to observe roughly 1 infected individual in a sample of 20, which is comparable with our observed data with no infected subjects.

Generally, we should be careful when using {_ysim} with Bayesian postestimation commands because it refers to all observations of the outcome variable. A better approach is to use a subset of observations, {_ysim[*numlist*]}, such as {_ysim[1/10]}. In our example, we have only one observation, so this specification is equivalent to specifying only the first observation, {_ysim[1]}.

◁

Example 3: Expressions of individual prediction results

We can compute posterior summaries for the expressions involving the individual values, {_ysim[#]}, where # refers to an observation. For instance, let's calculate the probability of observing 0 infected subjects in our sample of 20. Recall that our only observation records the number of observed infected subjects. We can estimate the probability that the outcome value is 0 as a proportion of 0 values of our simulated outcome in a sample of 10,000 MCMC replicates. We can do this by specifying the expression {_ysim[1]}==0 in bayesstats summary.

```
. bayesstats summary (prob0:{_ysim[1]}==0) using betabin_pred
Posterior summary statistics                       MCMC sample size =    10,000
       prob0 : _ysim1_1==0
```

	Mean	Std. dev.	MCSE	Median	Equal-tailed [95% cred. interval]	
prob0	.4479	.497303	.00708	0	0	1

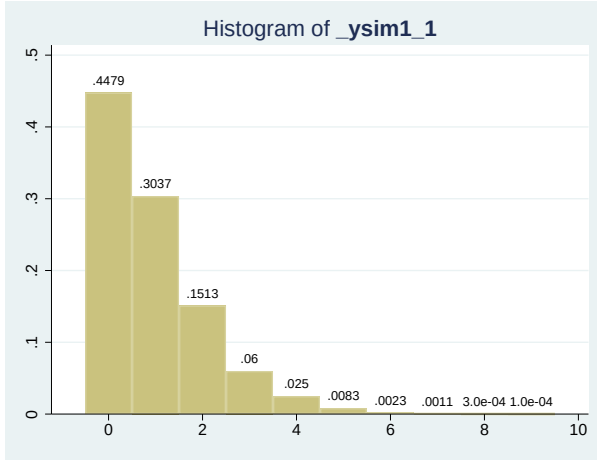
The posterior predictive mean (probability) for observing 0 infected subjects in the sample of 20 is 0.45, with a posterior predictive standard deviation of 0.5.

◁

Example 4: Visualizing prediction results

We can use graphical tools such as the histogram to summarize the posterior predictive distribution.

```
. bayesgraph histogram {_ysim[1]} using betabin_pred, discrete addlabels
```



The mass of the posterior predictive distribution for the number of infected subjects is concentrated on small numbers such as 0, 1, and 2 and thus agrees with what we observed in our sample.

Example 5: Posterior summaries of simulated outcomes

We can compute the posterior mean of the simulated outcome and save it in the current dataset as a new variable.

```
. bayespredict pmean, mean rseed(16)
Computing predictions ...
. summarize pmean
```

Variable	Obs	Mean	Std. dev.	Min	Max
pmean	1	.9526	.	.9526	.9526

The sample mean of `pmean` is an estimate of the posterior predictive mean of the outcome y and is the same as the one we obtained earlier by using `bayesstats summary`. Notice that we obtained the exact same values only because we used the same random-number seed, `rseed(16)`, with `bayespredict` when simulating the outcome `{_ysim}` and the posterior mean `pmean`.

If you need only posterior summaries of simulated outcomes, the above approach is preferable because it does not create a potentially large prediction dataset containing all MCMC replicates.

As the final step, we remove all the datasets created by `bayesmh` and `bayespredict` because we no longer need them, but you may choose to keep yours.

```
. erase betabin_mcmc.dta
. erase betabin_pred.dta
. erase betabin_pred.ster
```

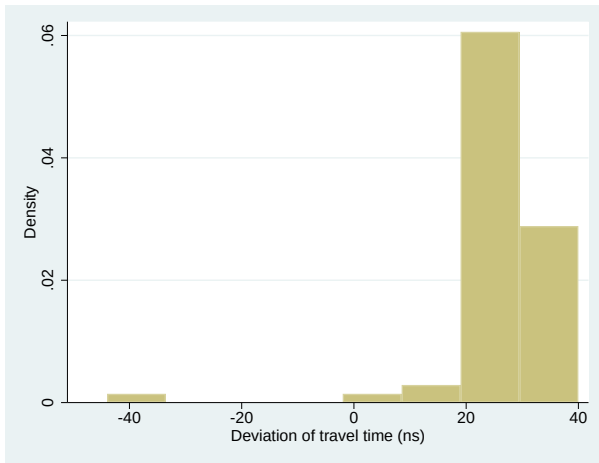
Posterior predictive inference

To illustrate posterior predictive checking, we adapt an example described in Gelman et al. (2014, sec. 6.3). The example analyzes the speed of light measurements from the experiment performed by Newcomb (1891). Newcomb measured the time (in nanoseconds) it takes for light to travel 7,442 meters. `splight.dta` contains 66 independent measurements of the deviance of the travel time from 24,800 nanoseconds in variable `timedev`.

```
. use https://www.stata-press.com/data/r17/splight
(Newcomb's speed of light measurements)
. describe
Contains data from https://www.stata-press.com/data/r17/splight.dta
 Observations:            66                  Newcomb's speed of light
                                                measurements
    Variables:             1                  22 Feb 2021 13:24
                                              (_dta has notes)
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
timedev         byte    %9.0g                 Deviation of travel time (ns)
-------------------------------------------------------------------------------
Sorted by:
```

Let's look at the distribution of the data.

```
. histogram timedev
(bin=8, start=-44, width=10.5)
```



The data have several extreme observations in the left tail—the smallest observed `timedev` is -44, which is more than 6 standard deviations smaller than the sample mean.

To demonstrate posterior predictive checking, Gelman et al. (2014) intentionally used a simplified model for `timedev`, a normal model with unknown mean μ and variance σ^2, which may not be a good fit given the presence of extreme observations. The authors chose a noninformative prior for the model parameters, $(\mu, \sigma^2) \sim 1/\sigma^2$, to achieve more objective analysis.

We fit the described model using bayesmh as follows:

```
. bayesmh timedev, likelihood(normal({sig2}))
> prior({timedev:_cons}, flat) prior({sig2}, jeffreys)
> mcmcsize(1000) rseed(16) saving(splight_mcmc)
Burn-in ...
Simulation ...
Model summary
------------------------------------------------------------------------------
Likelihood:
  timedev ~ normal({timedev:_cons},{sig2})
Priors:
  {timedev:_cons} ~ 1 (flat)
          {sig2} ~ jeffreys
------------------------------------------------------------------------------
Bayesian normal regression                      MCMC iterations  =      3,500
Random-walk Metropolis-Hastings sampling        Burn-in          =      2,500
                                                MCMC sample size =      1,000
                                                Number of obs    =         66
                                                Acceptance rate  =      .2128
                                                Efficiency:  min =       .104
                                                             avg =      .1123
Log marginal-likelihood = -249.39408                         max =      .1207
------------------------------------------------------------------------------
                |                                                Equal-tailed
                |      Mean   Std. dev.     MCSE     Median  [95% cred. interval]
----------------+-------------------------------------------------------------
timedev         |
          _cons |  26.40191   1.306144   .128102   26.42451   23.57925   28.71792
----------------+-------------------------------------------------------------
           sig2 |  118.8588   21.83563   1.98746   115.8515   81.03243   163.9617
------------------------------------------------------------------------------
file splight_mcmc.dta saved.
```

The described prior is modeled in bayesmh by specifying the flat prior for {timedev:_cons}, the mean parameter of the normal model, and the Jeffreys prior for variance {sig2}. We requested a small MCMC sample of only 1,000. We also specified the saving() option to save MCMC estimates of model parameters, which is required to use bayespredict or bayesreps.

bayesmh reports a 95% equal-tailed credible interval of [23.6, 28.7] for {timedev:_cons}. The true deviance of the travel time of light is known to be 33.0 nanoseconds and is outside the reported credible interval. Clearly, our model does not produce an accurate estimate for the speed of light. The question is, Can we detect the misfit without the knowledge of the true value? We explore the answers to this question in the following examples:

▷ Example 6: Goodness of fit using MCMC replicates of simulated outcomes

One way of checking goodness of fit is to compare the observed sample with the replication samples drawn from the posterior predictive distribution. Any systematic discrepancy between replicated and observed data will indicate misfit.

Let's start with visual inspection of the replicated data. We can use the bayesreps command to generate 20 MCMC replicates for the outcome timedev. Each replicate has 66 observations and is saved as a new variable in the dataset. We specify tdrep* as a variable stub for the replicate names.

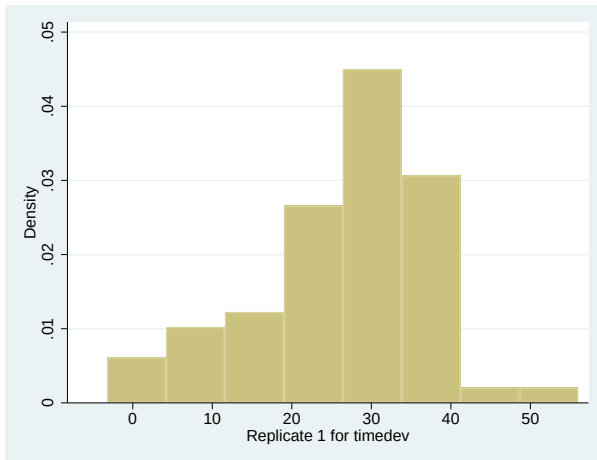
```
. bayesreps tdrep*, nreps(20) rseed(16)
Computing predictions ...
. summarize
```

Variable	Obs	Mean	Std. dev.	Min	Max
timedev	66	26.21212	10.74532	-44	40
tdrep1	66	26.10487	11.25766	-3.237112	56.03574
tdrep2	66	23.29179	11.30281	-12.58518	47.92589
tdrep3	66	26.21224	14.93573	-4.078057	61.37516
tdrep4	66	27.28245	11.60644	-2.502777	59.27184
tdrep5	66	27.74366	8.093924	5.70912	44.75622
tdrep6	66	26.15271	13.0279	-1.115488	53.42055
tdrep7	66	26.57665	10.11741	5.395408	46.71115
tdrep8	66	27.9395	12.06432	-4.903924	48.62425
tdrep9	66	25.54143	11.15095	1.560754	51.17381
tdrep10	66	28.11942	11.39326	1.364191	57.17214
tdrep11	66	24.18664	9.37403	7.49153	52.94038
tdrep12	66	25.87535	8.766691	10.5051	44.38683
tdrep13	66	27.49002	9.937486	4.89093	52.40338
tdrep14	66	26.17611	12.34034	-4.824428	51.16258
tdrep15	66	28.35187	10.58047	1.968471	50.73883
tdrep16	66	27.00237	11.44632	.7238956	52.77098
tdrep17	66	28.38859	11.1474	6.04494	63.34375
tdrep18	66	24.16652	9.289006	-.0226819	44.17939
tdrep19	66	24.9675	9.931602	6.675714	45.19432
tdrep20	66	27.69125	10.94969	1.289953	57.45961

The summary table shows that, compared with the observed data, the replicates have similar means and standard deviations but not the minimum and maximum values.

We can explore the entire distribution of a replicate. For example, we can produce the histogram for the first replicate and compare it with the earlier histogram of the observed data.

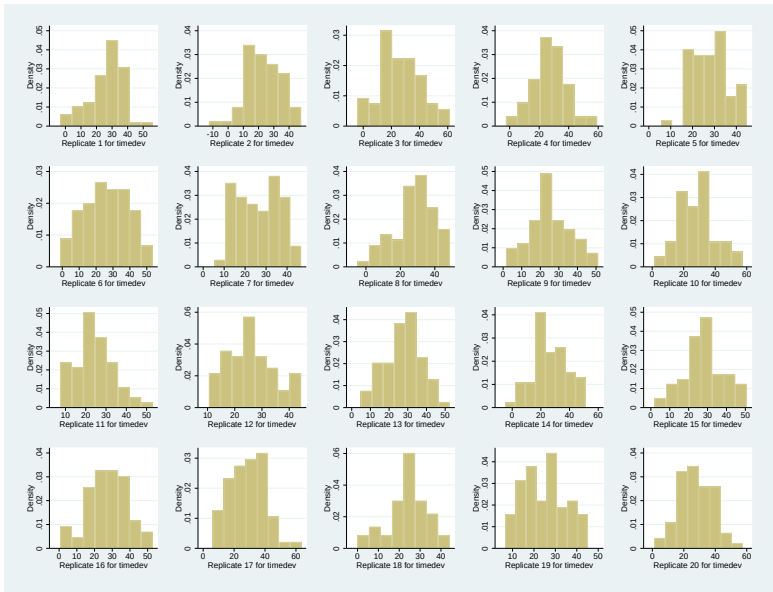
```
. histogram tdrep1
(bin=8, start=-3.237112, width=7.4091061)
```



The histograms look quite different. The replicate sample does not have the extreme negative values observed in the data.

With a few lines of code, we can produce histograms for all replicates and combine them on one graph.

```
. local histlist
. forvalues i = 1/20 {
  2.          quietly hist tdrep‘i’, name(hist‘i’) nodraw
  3.          local histlist ‘histlist’ hist‘i’
  4. }
. graph combine ‘histlist’
```



The histograms of all replicates look different from the observed data. The range for the replicated samples is about 0 to 50 with only a few negative values, which are smaller in magnitude than the negative values observed in the original data.

◁

▷ Example 7: Test statistics as scalar functions of simulated outcomes

Gelman et al. (2014) suggest to use the smallest observation to measure the discrepancy between the observed and replicated data. That is, to compare the smallest values in the replicated samples with Newcomb's smallest observation of −44.

In addition to simulating outcome values, as we demonstrated in example 1, we can use `bayespredict` to compute functions of simulated values that summarize the observations in a single statistic such as the minimum statistic. A function can be any Mata function that takes a column vector as an argument and returns a scalar. The result from `bayespredict` in this case is an MCMC sample of function values stored in the prediction dataset as a new variable.

Let's use `bayespredict` to produce an MCMC sample of the smallest observations (minimums) of the replicated data. Because we are not interested in individual observations, we can request that only

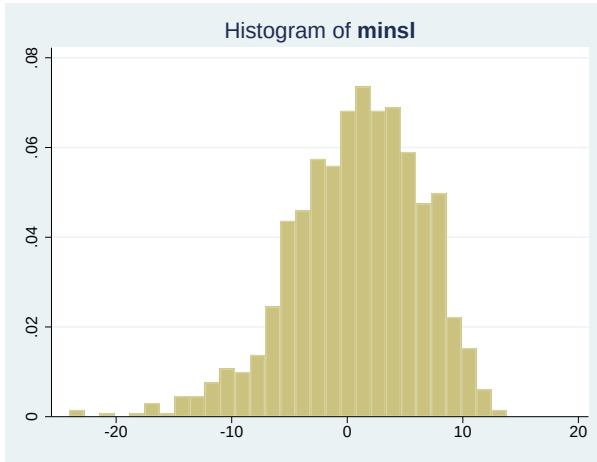
the smallest observation be simulated and stored by using the function specification `@min({_ysim})` with `bayespredict`.

```
. bayespredict (minsl:@min({_ysim})), saving(splight_pred) rseed(16)
Computing predictions ...
file splight_pred.dta saved.
file splight_pred.ster saved.
```

Per our specification, the command creates a new dataset, `splight_pred.dta`, that stores minimum statistics of the replicated data in the variable `minsl`. The prediction dataset has 1,000 observations, because 1,000 is the size of the MCMC sample simulated by `bayesmh`.

We can now use `{minsl}` within other Bayesian postestimation commands such as `bayesgraph` and `bayesstats summary` provided we supply the prediction dataset with the `using` specification. For example, let's draw the histogram of `{minsl}` using `bayesgraph histogram`.

```
. bayesgraph histogram {minsl} using splight_pred
```



The histogram provides the estimate of the posterior predictive distribution for the minimum statistic. The range of the histogram does not cover the observed minimum value of -44.

We can compare the posterior predictive distribution of the minimum statistic with the observed minimum value more formally by computing the posterior predictive p-value by using `bayesstats ppvalues`.

```
. bayesstats ppvalues {minsl} using splight_pred
Posterior predictive summary   MCMC sample size =     1,000
```

T	Mean	Std. dev.	E(T_obs)	P(T>=T_obs)
minsl	.8017725	5.590955	-44	1

```
Note: P(T>=T_obs) close to 0 or 1 indicates lack of fit.
```

The output table shows the posterior mean and standard deviation of `{minsl}`, the observed minimum value, -44, and the estimated posterior predictive p-value. The last is the probability that the replicated smallest value be greater or equal to the observed one. For a well-fitting model, the posterior predictive p-value should, ideally, be close to 0.5, although values between 0.05 and 0.95 are often considered acceptable in the literature (Gelman et al. 2014, 150). In our example, its estimate is essentially 1,

which indicates a strong misfit of the specified normal model. Therefore, if modeling of the tails of the outcome distribution is important, we should reconsider the normal likelihood model and find a better alternative.

◁

▷ Example 8: Test quantities via user-defined Stata programs

It is not sufficient to assess goodness of fit by examining just one test statistic. Different test statistics capture different aspects of the data. Which statistic to use depends on the research problem and the data characteristics you wish to account for. Generally, as pointed out by Gelman et al. (2014), for noninformative priors, sufficient statistics such as a sample mean and variance may not be good choices for checking model fit because they are typically modeled directly by the parameters of the likelihood function.

We demonstrated that our model does not model the minimum statistic well. Let's consider another aspect of `timedev`: symmetry with respect to the mean μ.

Following Gelman et al. (2014), we define the following test quantity to measure asymmetry,

$$T(\texttt{timedev}, \mu) = |\texttt{timedev}_{(61)} - \mu| - |\texttt{timedev}_{(6)} - \mu|$$

where $\texttt{timedev}_{(a)}$ defines the ath ordered value of `timedev` and $(\texttt{timedev}_{(6)}, \texttt{timedev}_{(61)})$ represents about 90% of the distribution of `timedev`.

There is no predefined computation for the above statistic, so we need to write our own. For statistics that depend only on simulated outcome values, expected values, and residuals, we can write our own Mata functions or Stata programs. Mata functions are generally faster. For statistics that directly use model parameters, writing a Stata program is our only choice. Because the calculation of $T(\texttt{timedev}, \mu)$ involves a model parameter, μ, we must write a Stata program to calculate this statistic. Let's call our program `symstatprog`.

```
program symstatprog
        version 17.0
        args symout ysim
        tempname mu
        scalar ‘mu’ = $BAYESPR_theta[1,1]
        sort ‘ysim’
        scalar ‘symout’ = abs(‘ysim’[61]-‘mu’)-abs(‘ysim’[6]-‘mu’)
end
```

The program has two input arguments, `symout` and `ysim`. The local macro `symout` contains the name of a temporary scalar for storing the final result. The local macro `ysim` contains the name of a temporary variable that stores the simulated outcome values of `timedev`. The global macro `$BAYESPR_theta` contains the name of a temporary matrix (row vector) that stores the current values of simulated model parameters, which are μ and σ^2 in our example. The parameters are stored in the same order they are displayed by `bayesmh`. Thus, in our example, the first element of this matrix corresponds to the mean, μ. We use the earlier definition to compute the asymmetry test quantity and store it in the scalar `‘symout’`.

We now call `bayespredict` to use the `symstatprog` program to compute the asymmetry test quantity for each set of simulated model parameters and label the prediction results as `symstat`. We replace our previously generated prediction dataset, `splight_pred.dta`, with these new prediction results.

```
. bayespredict (symstat:@symstatprog {_ysim}), saving(splight_pred, replace)
> rseed(16)
Computing predictions ...
file splight_pred.dta saved.
file splight_pred.ster saved.
```

We can use `bayesstats ppvalues` to test the goodness of fit for $T(\texttt{timedev}, \mu)$.

```
. bayesstats ppvalues {symstat} using splight_pred
Posterior predictive summary   MCMC sample size =     1,000
```

T	Mean	Std. dev.	E(T_obs)	P(T>=T_obs)
symstat	.0953002	3.476211	3.196186	.235

```
Note: P(T>=T_obs) close to 0 or 1 indicates lack of fit.
```

The posterior predictive p-value is estimated to be 0.235 and does not suggest model misfit with respect to $T(\texttt{timedev}, \mu)$.

◁

▷ Example 9: Working with a prediction dataset

Sometimes, we may need to access the prediction results. For example, Gelman et al. (2014) provide a visual representation of the posterior predictive p-value by plotting the observed values of the asymmetry test quantity, $T(\texttt{timedev}, \mu)$, versus the replicated values, $T(\texttt{timedev}^{\text{rep}}, \mu)$. We can reproduce this graph as follows.

We start by loading the prediction dataset that contains our prediction results.

```
. use splight_pred, clear
. describe
Contains data from splight_pred.dta
 Observations:         1,000
    Variables:             5                  2 Jun 2021 18:55
```

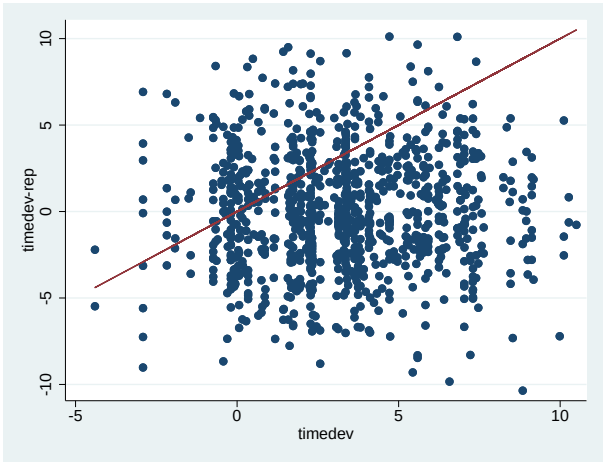
Variable name	Storage type	Display format	Value label	Variable label
_chain	int	%8.0g		Chain identifier
_index	int	%8.0g		Iteration number
symstat	double	%10.0g		symstatprog {_ysim1}
_obs_symstat	double	%10.0g		Observed symstatprog {_ysim1}
_frequency	byte	%8.0g		Frequency weight

```
Sorted by:
```

Similarly to the MCMC simulation dataset, variables `_chain` and `_index` record chain and index identifiers. Variable `symstat` contains the values for $T(\texttt{timedev}^{\text{rep}}, \mu)$, and variable `_obs_symstat` contains the values for $T(\texttt{timedev}, \mu)$. For consistency with the simulation dataset, the prediction dataset also contains the `_frequency` variable, but it is always one in the prediction dataset.

To visualize the posterior predictive p-value, we draw the scatterplot of `symstat` versus `_obs_symstat` overlaid with the diagonal line for `_obs_symstat` as the reference line.

```
. scatter symstat _obs_symstat || line _obs_symstat _obs_symstat,
> xtitle("timedev") ytitle("timedev-rep") legend(off)
```



The estimated posterior predictive p-value is the proportion of points above the diagonal line.

In conclusion, although the normal model describes well the symmetry of the observed measurements, it fails to capture some of the smaller observations. It is possible that the experimental procedure was susceptible to aberrant measurements and a different model is needed to reflect this.

Out-of-sample prediction

This section illustrates how `bayespredict` can be used as a classifier for binary outcomes.

Example 10: Out-of-sample classification using predictive posterior means

We consider `titanic800.dta`, which contains the information of 800 passengers, who were on board the ocean liner *Titanic* when it sank. The dataset is a subset from a larger dataset published by Dawson (1995) at https://www.amstat.org/publications/jse/v3n3/datasets.dawson.html.

```
. use https://www.stata-press.com/data/r17/titanic800, clear
(Titanic passenger survival (Extract))
. describe
Contains data from https://www.stata-press.com/data/r17/titanic800.dta
 Observations:           800                  Titanic passenger survival
                                                (Extract)
    Variables:             4                  22 Feb 2021 13:24
                                              (_dta has notes)
-------------------------------------------------------------------------------
Variable      Storage   Display    Value
    name         type    format    label      Variable label
-------------------------------------------------------------------------------
class           byte    %9.0g      class      Class
adult           byte    %9.0g      age        Adult
male            byte    %9.0g      sex        Male
survived        byte    %9.0g      survived   Survived
-------------------------------------------------------------------------------
Sorted by:
```

The binary variable `survived` records whether a passenger survived (`survived` = 1) or not (`survived` = 0). Passenger characteristics include the cabin type and class membership, `class` (first, second, third, or crew); the sex, `male`; and whether the passenger was an adult or a child, `adult`.

For illustration, we consider a simple logistic regression of `survived` on the categorical predictor `class` and binary predictors `male` and `adult`.

First, we randomly split the data into training and test subsamples. We use `splitsample` ([D] **splitsample**) to generate a variable, `sample`, that assigns 50% of the data to the training subsample (`sample` = 1) and the other 50% to the test subsample (`sample` = 2).

```
. splitsample, generate(sample) rseed(12345)
```

Second, we fit a Bayesian logistic regression using the training subsample of 400 passengers. We apply a Cauchy$(0, 1)$ prior distribution for the coefficients. As a prerequisite for computing Bayesian predictions, we save the MCMC sample in `titanic_mcmc.dta`.

```
. bayesmh survived i.male i.adult ib1.class if sample==1, likelihood(logit)
> prior({survived:}, cauchy(0, 1)) saving(titanic_mcmc) rseed(16)
Burn-in ...
Simulation ...
Model summary
------------------------------------------------------------------------------
Likelihood:
  survived ~ logit(xb_survived)
Prior:
  {survived:1.male 1.adult i.class _cons} ~ cauchy(0,1)                   (1)
------------------------------------------------------------------------------
(1) Parameters are elements of the linear form xb_survived.
Bayesian logistic regression                    MCMC iterations  =     12,500
Random-walk Metropolis-Hastings sampling        Burn-in          =      2,500
                                                MCMC sample size =     10,000
                                                Number of obs    =        400
                                                Acceptance rate  =      .2054
                                                Efficiency:  min =     .02887
                                                             avg =     .04189
Log marginal-likelihood = -211.35694                         max =     .05692
------------------------------------------------------------------------------
                                                              Equal-tailed
    survived |      Mean   Std. dev.      MCSE     Median  [95% cred. interval]
-------------+----------------------------------------------------------------
        male |
        male |  -2.490095   .3330118   .019599  -2.498318   -3.13982  -1.844389
             |
       adult |
       adult |  -.5916052   .4910491   .024348  -.5666577  -1.551435   .3630116
             |
       class |
        crew |  -.6376593    .389797    .01675  -.6224151  -1.435266    .118966
      second |  -.5605325   .4214846   .017667  -.5507987  -1.423903   .2334895
       third |  -1.103689   .4064315   .021184  -1.106785  -1.923915  -.3518597
             |
       _cons |   2.386679   .6342651   .034439   2.384843   1.188944   3.692627
------------------------------------------------------------------------------
file titanic_mcmc.dta saved.
```

All coefficients are negative, which means they are associated with lower survival probabilities compared with their respective baselines. For instance, adults were less likely to survive than children, and crew members and second- and third-class passengers were less likely to survive than the first-class passengers. The male passengers on board *Titanic* were especially unfortunate—the posterior mean estimate for the coefficient on `male` is -2.5 with a 95% credible interval of $[-3.1, -1.8]$.

Let's now compute out-of-sample predictions for the test subsample of the other 400 passengers. We use `bayespredict` with the `mean` option to calculate the posterior means of the simulated outcome for these passengers and store them as a new variable, `pmean`, in the current dataset.

```
. bayespredict pmean if sample==2, mean dots(100, every(1000)) rseed(16)
Computing predictions 10000 .........1000.........2000.........3000.........
> 4000.........5000.........6000.........7000.........8000.........9000.........
> 10000 done
```

The posterior means estimate the survival probabilities of the passengers and are, in fact, the optimal predictors with respect to the mean squared error (MSE). Let's compute MSE for `pmean` to assess prediction accuracy of the model.

```
. generate err2 = (survived-pmean)^2
(400 missing values generated)
. summarize err2 if sample==2
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
        err2 |        400    .1740713    .2328064   .0187416    .741321
```

Our model achieves an MSE of 0.17, but this number is difficult to interpret on its own, without any reference models.

Let's compute the prediction accuracy of our model or how well our model predicted the outcome in the test subsample. We generate a new variable, `survived_logit`, to contain the binary outcome predicted from our Bayesian logistic model. We assign the predicted outcome to be 1 if `pmean` is greater than 0.5, and 0 otherwise. We then estimate the prediction accuracy as the proportion of matches between the observed `survived` and the predicted `survived_logit` in the test subsample.

```
. generate survived_logit = (pmean>0.5)
. generate pacc = (survived==survived_logit)
. summarize pacc if sample==2
    Variable |        Obs        Mean    Std. dev.       Min        Max
-------------+---------------------------------------------------------
        pacc |        400         .76     .427618         0          1
```

The prediction accuracy of our simple logistic model is about 0.76, which is not that high. Thus, a better prediction model should be considered for these data.

◁

One-step-ahead Bayesian forecast after Bayesian VAR

After fitting Bayesian VAR models using the `bayes: var` command, you can use `bayespredict` to compute Bayesian forecasts; see example 10 in [BAYES] **bayes: var**.

Stored results

`bayespredict` stores the following in an estimation file, *filename*`.ster`, where *filename* is specified in the `saving(`*filename*`)` option.

Scalars

`e(N)`	number of observations
`e(nchains)`	number of MCMC chains
`e(mcmcsize)`	MCMC sample size

Macros

`e(cmd)`	`bayespredict`
`e(est_cmd)`	`bayesmh`
`e(cmdline)`	command as typed
`e(est_cmdline)`	estimation command as typed
`e(predfile)`	file containing prediction results
`e(mcmcfile)`	file containing simulation results
`e(predynames)`	names of simulated outcome observations, `_ysim#_#`
`e(predfnames)`	names of specified functions and programs
`e(predrngstate#)`	random-number state for #th chain for prediction
`e(rngstate)`	random-number state for simulation (only with single chain)
`e(rngstate#)`	random-number state for #th chain for simulation (only with `nchains()`)

Methods and formulas

Methods and formulas are presented under the following headings:

Posterior predictive distribution

Recall from *Overview of Bayesian predictions* that the posterior predictive distribution of new data $\mathbf{y}^{\text{new}}$ given observed data $\mathbf{y}^{\text{obs}}$ is

$$\begin{aligned} p(\mathbf{y}^{\text{new}}|\mathbf{y}^{\text{obs}}) &= \int p(\mathbf{y}^{\text{new}}, \boldsymbol{\theta}|\mathbf{y}^{\text{obs}}) d\boldsymbol{\theta} \\ &= \int p(\mathbf{y}^{\text{new}}|\mathbf{y}^{\text{obs}}, \boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{y}^{\text{obs}}) d\boldsymbol{\theta} \\ &= \int p(\mathbf{y}^{\text{new}}|\boldsymbol{\theta}) p(\boldsymbol{\theta}|\mathbf{y}^{\text{obs}}) d\boldsymbol{\theta} \end{aligned} \tag{3}$$

where we used the assumption of independence between $\mathbf{y}^{\text{new}}$ and $\mathbf{y}^{\text{obs}}$ given $\boldsymbol{\theta}$ to arrive at the final expression.

Simulated outcomes, $\mathbf{y}^{\text{sim}}$, are the outcome values simulated from the posterior predictive distribution (3).

In a regression setting, posterior predictive distribution (3) also depends on the covariate data,

$$p(\mathbf{y}^{\text{new}}|\mathbf{y}^{\text{obs}}, X^{\text{new}}) = \int p(\mathbf{y}^{\text{new}}|\boldsymbol{\theta}, X^{\text{new}}) p(\boldsymbol{\theta}|\mathbf{y}^{\text{obs}}, X^{\text{obs}}) d\boldsymbol{\theta} \tag{4}$$

where X^{new} is the data matrix containing new covariate values and X^{obs} is the data matrix containing observed covariate values used to fit the model.

The concept of replicated outcomes or replicated data, $\mathbf{y}^{\text{rep}}$, arises in a regression setting when the data matrix used to generate new outcome values is the same as the observed data matrix used to fit the Bayesian model. That is,

$$p(\mathbf{y}^{\text{rep}}|\mathbf{y}^{\text{obs}}, X^{\text{obs}}) = \int p(\mathbf{y}^{\text{rep}}|\boldsymbol{\theta}, X^{\text{obs}}) p(\boldsymbol{\theta}|\mathbf{y}^{\text{obs}}, X^{\text{obs}}) d\boldsymbol{\theta} \tag{5}$$

In a regression setting, we use a general definition for the simulated outcome, $\mathbf{y}^{\text{sim}}$, as one generated either from (4) or (5).

Test quantities and test statistics are commonly used to check goodness of fit of a Bayesian model. A test quantity, $T_q(\mathbf{y}^{\text{rep}}, \boldsymbol{\theta})$, is a scalar function of replicated data $\mathbf{y}^{\text{rep}}$ and model parameters $\boldsymbol{\theta}$. A test statistic, $T_s(\mathbf{y}^{\text{rep}})$, is a scalar function that depends only on the replicated data $\mathbf{y}^{\text{rep}}$. If the model fits the data well, $T_q(\mathbf{y}^{\text{rep}}, \boldsymbol{\theta})$ should be close to $T_q(\mathbf{y}^{\text{obs}}, \boldsymbol{\theta})$, and, similarly, $T_s(\mathbf{y}^{\text{rep}})$ should be close to $T_s(\mathbf{y}^{\text{obs}})$.

MCMC sampling from posterior predictive distribution

Like the posterior distribution of model parameters, posterior predictive distributions (3), (4), and (5) usually do not have closed forms and must be approximated. In what follows, we will concentrate on the more general posterior predictive distribution $p(\mathbf{y}^{\text{new}}|\mathbf{y}^{\text{obs}}, X^{\text{new}})$, but the same principles apply to the other distributions by removing conditioning on covariate data in case of (3) and by replacing X^{new} with X^{obs} in case of (5).

The goal of Bayesian prediction is to simulate data from $p(\mathbf{y}^{\text{new}}|\mathbf{y}^{\text{obs}}, X^{\text{new}})$. Formula (4) underlies the following two-step iterative process for obtaining simulated outcomes from $p(\mathbf{y}^{\text{new}}|\mathbf{y}^{\text{obs}}, X^{\text{new}})$.

1. Draw a realization of model parameters, $\boldsymbol{\theta}^{\star}$, from their posterior distribution, $p(\boldsymbol{\theta}|\mathbf{y}^{\text{obs}}, X^{\text{obs}})$.
2. Generate $\mathbf{y}^{\text{sim}}$ from $p(\mathbf{y}^{\text{new}}|\boldsymbol{\theta}^{\star}, X^{\text{new}})$, the data distribution (likelihood) conditional on the parameters obtained in step 1.

Steps 1 and 2 are repeated to produce an MCMC sample of simulated outcomes, $(\mathbf{y}^{\text{sim},1}, \mathbf{y}^{\text{sim},2}, \ldots, \mathbf{y}^{\text{sim},\text{T}})$, where T is the MCMC sample size. We can use this sample to estimate the posterior predictive distribution.

For step 1, `bayespredict` uses the MCMC sample of model parameters as produced by the `bayesmh` command. The main computation of `bayespredict` is the simulation of the outcome values from the respective likelihood model for each set of simulated model parameters from the MCMC sample. For an outcome variable with n observations, the result of a Bayesian prediction is a dataset containing T observations and n columns.

A function of simulated values is computed as follows: $\{f(\mathbf{y}^{\text{sim},1}), f(\mathbf{y}^{\text{sim},2}), \ldots, f(\mathbf{y}^{\text{sim},\text{T}})\}$, where $f(\cdot)$ is a function that operates on a column vector and returns a scalar. The resulting prediction dataset will contain a variable with T observations.

For a test statistic $T_s(\mathbf{y}^{\text{rep}})$, the following simulated sample is produced:

$\{T_s(\mathbf{y}^{\text{rep},1}), T_s(\mathbf{y}^{\text{rep},2}), \ldots, T_s(\mathbf{y}^{\text{rep},\text{T}})\}$. For a well-fitting model, the distribution of this sample should be concentrated around $T_s(\mathbf{y}^{\text{obs}})$.

For a test quantity $T_q(\mathbf{y}^{\text{rep}}, \boldsymbol{\theta})$, the following simulated sample is produced:

$\{T_q(\mathbf{y}^{\text{rep},1}, \boldsymbol{\theta}^1), T_q(\mathbf{y}^{\text{rep},2}, \boldsymbol{\theta}^2), \ldots, T_q(\mathbf{y}^{\text{rep},\text{T}}, \boldsymbol{\theta}^T)\}$. For a well-fitting model, the distribution of this sample should be close to the distribution of $\{T_q(\mathbf{y}^{\text{obs}}, \boldsymbol{\theta}^1), T_q(\mathbf{y}^{\text{obs}}, \boldsymbol{\theta}^2), \ldots, T_q(\mathbf{y}^{\text{obs}}, \boldsymbol{\theta}^T)\}$.

Residuals and expected values

Consider simulated outcome values $\mathbf{y}_i^{\text{sim}}$ for an observation $i = 1, 2, \ldots, n$, where $\mathbf{y}_i^{\text{sim}} = (y_i^{\text{sim},1}, y_i^{\text{sim},2}, \ldots, y_i^{\text{sim},\text{T}})^T$. Let $\widehat{\boldsymbol{\mu}}_i = (\widehat{\mu}_i^1, \widehat{\mu}_i^2, \ldots, \widehat{\mu}_i^{\text{T}})^T$, where $\widehat{\mu}_i^t = E(y_i|\mathbf{x}_i, \boldsymbol{\theta}^t)$ is the estimated expected value of y_i given covariate vector $\mathbf{x}_i$ and simulated parameters $\boldsymbol{\theta}^t$, $t = 1, 2, \ldots, T$. Let $\mathbf{r}_i^{\text{sim}} = (r_i^{\text{sim},1}, r_i^{\text{sim},2}, \ldots, r_i^{\text{sim},\text{T}})^T$ be simulated residuals for an observation i.

Simulated residuals are then defined as

$$\mathbf{r}_i^{\text{sim}} = \mathbf{y}_i^{\text{sim}} - \widehat{\boldsymbol{\mu}_i}$$

Within `bayespredict`, you refer to $\mathbf{y}_i^{\text{sim}}$ as `{_ysim_`i`}`, $\mathbf{r}_i^{\text{sim}}$ as `{_resid_`i`}`, and $\widehat{\boldsymbol{\mu}_i}$ as `{_mu_`i`}`. You can also use `{_ysim}`, `{_resid}`, and `{_mu}` to refer to all observations at once. With multiple outcomes, the above specifications correspond to the first outcome variable. For the #th outcome variable, use `{_ysim#_`i`}`, `{_resid#_`i`}`, `{_mu#_`i`}`, `{_ysim#}`, `{_resid#}`, and `{_mu#}`, respectively.

Below are the definitions of $\widehat{\mu}_i^t$ for the likelihood models supported by `bayesmh`.

1. **Normal regression**: $\widehat{\mu}_i^t = \mathbf{x}_i\boldsymbol{\beta}^t$.
2. **t-regression**: $\widehat{\mu}_i^t = \mathbf{x}_i\boldsymbol{\beta}^t$.
3. **Lognormal regression**: $\widehat{\mu}_i^t = \exp(\mathbf{x}_i\boldsymbol{\beta}^t)$.
4. **Exponential regression**: $\widehat{\mu}_i^t = \exp(\mathbf{x}_i\boldsymbol{\beta}^t)$.
5. **Probit regression**: $\widehat{\mu}_i^t = \Phi(\mathbf{x}_i\boldsymbol{\beta}^t)$.
6. **Logistic regression**: $\widehat{\mu}_i^t = \text{invlogit}(\mathbf{x}_i\boldsymbol{\beta}^t)$.
7. **Binomial regression**: $\widehat{\mu}_i^t = n_{\text{trials}} \times \text{invlogit}(\mathbf{x}_i\boldsymbol{\beta}^t)$, where n_{trials} is the number of trials in binomial regression.
8. **Ordered probit regression**: `{_resid}` and `{_mu}` not supported.
9. **Ordered logistic regression**: `{_resid}` and `{_mu}` not supported.
10. **Poisson regression**: $\widehat{\mu}_i^t = \exp(\mathbf{x}_i\boldsymbol{\beta}^t)$.

Next are the definitions of $\widehat{\mu}_i^t$ for the distribution models `dexponential(`*beta*`)`, `dbernoulli(`*p*`)`, `dbinomial(`n_{trials}`,`*p*`)`, and `dpoisson(`*mu*`)`.

11. **Exponential distribution**: $\widehat{\mu}_i^t = \beta^t$.
12. **Bernoulli distribution**: $\widehat{\mu}_i^t = p^t$.
13. **Binomial distribution**: $\widehat{\mu}_i^t = n_{\text{trials}}p^t$.
14. **Poisson distribution**: $\widehat{\mu}_i^t = \mu^t$.

Typically, the expected values for the distribution models will be constant over observations unless the distribution parameters vary over the observations.

Raw residuals, $\mathbf{r}_i^{\text{sim}}$, may not always be the most appropriate for diagnostic purposes. For example, Pearson residuals are better suited for discrete outcome models such as binomial and Poisson regressions.

References

Dawson, R. J. M. 1995. The "Unusual Episode" data revisited. *Journal of Statistics Education* 3(3): 1–9. https://doi.org/10.1080/10691898.1995.11910499.

Gelman, A., J. B. Carlin, H. S. Stern, D. B. Dunson, A. Vehtari, and D. B. Rubin. 2014. *Bayesian Data Analysis*. 3rd ed. Boca Raton, FL: Chapman & Hall/CRC.

Gelman, A., X.-L. Meng, and H. S. Stern. 1996. Posterior predictive assessment of model fitness via realized discrepancies. *Statistica Sanica* 6: 733–760.

Gelman, A., and D. B. Rubin. 1992. Inference from iterative simulation using multiple sequences. *Statistical Science* 7: 457–472. https://doi.org/10.1214/ss/1177011136.

Hoff, P. D. 2009. *A First Course in Bayesian Statistical Methods*. New York: Springer.

Meng, X.-L. 1994. Multiple-imputation inferences with uncongenial sources of input (with discussion). *Statistical Science* 9: 538–573. https://doi.org/10.1214/ss/1177010269.

Newcomb, S. 1891. Measures of the velocity of light made under the direction of the Secretary of the Navy during the years 1880–1882. *Astronomical Papers* 2: 107–229.

Tsui, K.-W., and S. Weerahandi. 1989. Generalized p-values in significance testing of hypotheses in the presence of nuisance parameters. *Journal of the American Statistical Association* 84: 602–607. https://doi.org/10.2307/2289949.

West, M. 1986. Bayesian model monitoring. *Journal of the Royal Statistical Society, Series B* 48: 70–78. https://doi.org/10.1111/j.2517-6161.1986.tb01391.x.

Also see

[BAYES] **bayesmh** — Bayesian models using Metropolis–Hastings algorithm

[BAYES] **bayesgraph** — Graphical summaries and convergence diagnostics

[BAYES] **Bayesian postestimation** — Postestimation tools for bayesmh and the bayes prefix

[BAYES] **bayesstats ess** — Effective sample sizes and related statistics

[BAYES] **bayesstats ppvalues** — Bayesian predictive p-values and other predictive summaries

[BAYES] **bayesstats summary** — Bayesian summary statistics

[BAYES] **bayestest interval** — Interval hypothesis testing